



EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

# Data Management Division

DFS Software

FTU

FITS Translation Utility

User Manual

Doc.No. VLT-MAN-ESO-19000-2050

Issue 8

Date 27/01/2003

Prepared..... J.Knudstrup ..... 27/01/2003 .....  
Name Date Signature

Approved..... A.Wicenec .....  
Name Date Signature

Released ..... M.Peron/P.Quinn .....  
Name Date Signature



**Change Record**

Issue/Rev.	Date	Section/Page Affected	Reason/Initiation/Document/Remarks
1.0	07/02/2000	All	First preparation.
1.1	11/04/2000	All	Introduced suggestions proposed due to testing (G.Mekiffer, K.Haggouchi).
1.2	07/07/2000	Added section about valid characters.  Added description of the FILE.PATH.* keywords.	DFS00730  Requested by A.Wicenec.
1.4	02/02/2001	Minor updates	Comments from G.Mekiffer.
5	22/02/2001	Added information about FTU FTT Python API.	FTT Python API implemented on request from A.Wicenec.
6	21/11/2001	Mentioned ftuGenMoonPars, possible to add COMMENTS via FTT.	Implemented ftuGenMoonPars + added possibility for adding COMMENT keywords via the FTTs.
7	15/02/2002	Section 2.15.	Added description of new keyword: FILE.SPLIT.HDUS.
8	27/01/2003	Section 2.9.	Added description of External Conversion Function extended command interface.



## TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>7</b>
1.1	PURPOSE .....	7
1.2	REFERENCE DOCUMENTS .....	7
1.3	ABBREVIATIONS AND ACRONYMS .....	7
1.4	GLOSSARY .....	8
1.5	STYLISTIC CONVENTION .....	8
<b>2</b>	<b>OVERVIEW &amp; USAGE</b>	<b>9</b>
2.1	BASIC FUNCTIONING .....	9
2.2	VALID CHARACTERS FOR FILE- AND KEYWORD NAMES .....	10
2.3	KEYWORD MANIPULATION ACTIONS .....	10
2.4	FILE RENAMING .....	11
2.5	EXTENDED FITS FILE SPLITTING .....	11
2.6	EXTRACTING FITS HEADERS .....	11
2.7	SPECIFYING OUTPUT FILE TARGET DIRECTORY .....	11
2.8	CHANGES INTRODUCED TO FITS HEADERS .....	11
2.9	VALUE SOURCES/CONVERSION FUNCTIONS .....	12
2.10	TYPE CONVERSION .....	14
2.11	IMAGE STATISTICS .....	15
2.12	CALCULATION OF CHECKSUM .....	15
2.13	LOGGING ACTIONS IN A LOG FILE .....	15
2.14	THE COMMAND LINE PARAMETERS .....	16
2.15	THE FITS TRANSLATION TABLE .....	17
2.16	CAUTIONS .....	22
2.16.1	'Problematic' Keywords .....	22
2.16.2	Keyword Definitions for Rename/Copy Actions .....	22
<b>3</b>	<b>FTU FTT Python API</b>	<b>23</b>
3.1	CLASSES/MODULES IN THE FTT API .....	23
3.2	EXAMPLE OF A Python FTT .....	24



# 1 INTRODUCTION

## 1.1 PURPOSE

The FITS Translation Utility is used to manipulate FITS files. This implies mainly changing parameters contained in the headers of these files. This is needed for instance to correct for illegal or missing parameter values in the files being produced by the instruments. The tool can also be used by users who want to translate the headers of FITS files into a format more appropriate for a certain context.

This document is the user manual for the FITS Translation Utility. It describes in detail how to use this tool for translating FITS files.

To obtain the latest releases of the tool and other information, consult the FTU WEB site:

**<http://archive.eso.org/ftu>**

## 1.2 REFERENCE DOCUMENTS

The following documents are referenced in this document:

- [1] Definition of the Flexible Image Transport System (FITS), March 29, 1999, NOST 100-2.0, <http://fits.gsfc.nasa.gov>.
- [2] Data Interface Control Document, November 25, 1997, GEN-SPE-ESO-19400-794/1.1/0, <http://www.eso.org/dicb>.

## 1.3 ABBREVIATIONS AND ACRONYMS

The following abbreviations and acronyms are used in this document:

ECF	External Conversion Function
ICF	Internal Conversion Function
ISO	International Standardisation Organisation
FTT	FITS Translation Table
FTU	FITS Translation Utility
SW	Software
TBC	To Be Confirmed
TBD	To Be Defined
VLT	Very Large Telescope
WS	Workstation

## 1.4 GLOSSARY

### *Concatenation File Renaming*

Scheme for renaming FITS files whereby a number of values are concatenated with an underscore character, and an index appended if needed to make the name unique.

### *(Internal/External) Conversion Function*

The Conversion Functions are used to convert a value or a number of values into a single value before assigning it to the keyword which is written in a FITS header.

### *FITS Translation Table*

ASCII file in PAF format which defines the Translation Actions the FTU should carry out on a FITS file.

### *Prefix File Renaming*

Scheme for renaming FITS files whereby a static prefix is appended a unique, four digit index which is allocated automatically by the FTU.

### *Translation Action*

Indicates an action performed by the FTU specified in the FTT.

### *Value Source*

Indicates a location from where the FTU can retrieve a value used as basis for instance for the value of a keyword, possibly in conjunction with a number of other values from other Value Sources. Possible Value Sources are a keyword, a constant and an Internal/External Conversion Function.

## 1.5 STYLISTIC CONVENTION

The following styles are used:

### **bold**

in the text, for commands, filenames, pre/suffixes as they have to be typed.

### *italic*

in the text, for parts that have to be substituted with the real content before typing.

### `teletype`

for examples.

### `<name>`

in the examples, for parts that have to be substituted with the real content before typing.

**bold** and *italic* are also used to highlight words.



The output files are stored in the same directory as the input files.

The headers in a FITS file are addressed with an integer number, whereby the primary header is number 0 and the first header extension has number 1, and so forth.

## 2.2 VALID CHARACTERS FOR FILE- AND KEYWORD NAMES

The character set considered as valid for building filenames and keyword names is all alpha-numeric characters and in addition underscore (" \_"), dash ("-") and colon (":"). I.e., "0, ..., 9, a, ..., z, A, ..., Z, \_, -, :".

There is no strict checking of whether the names conform to this scheme, but it is recommended to apply this rule. However, when filenames are generated special characters will be substituted with an underscore character automatically by the FTU.

## 2.3 KEYWORD MANIPULATION ACTIONS

The following translation actions can be performed on keywords in FITS headers:

<b>ADD</b>	<p>The Add Action allows to add new keywords in a FITS header. The value for the new keyword can come from one of the following sources: 1) Another key, 2) An Internal or External Conversion Function, 3) A constant.</p> <p>The header in which to add the key must be given. A Conversion Function to apply to the input values before assigning it to the key can be defined.</p>
<b>CHANGE</b>	<p>The Change Action is used to change the value of an existing keyword. The value is handled as for the Add Action.</p>
<b>RENAME</b>	<p>The Rename Action is used to rename an existing key. The source and target keyword names must be defined as well as the header.</p>
<b>MOVE</b>	<p>The Move Action is used to move a FITS card (keyword record) from one header to another. Apart from the keyword, the source and target headers must be specified. A Conversion Function can be applied to the original value before writing the key in the target header.</p>
<b>COPY</b>	<p>The Copy Action is used to copy a keyword, possibly from one header to another. The source keyword/source header must be specified as well as the target key/target header. A Conversion Function can be specified.</p> <p>When copying within the same header, there will still be only one occurrence of the key as the FTU does not allow repeated keywords.</p>
<b>REMOVE</b>	<p>The Remove Action is used to remove a keyword or a list of keywords from a header. The keywords to be removed must be specified as a comma separated list. It is possible to use wild cards. The header in which to remove the keywords must be specified.</p>

The actual tags to use in the FTT are described in section 2.15.

## 2.4 FILE RENAMING

It can be requested to rename the files being translated. The renaming is done by specifying a prefix to which one or more values extracted from specified Value Sources can be appended. The prefix can be of zero length (specified as ""). The values extracted from the Value Sources are appended with an underscore character ("\_"). If a file with the generated name already is found in the target directory, an index is automatically added.

The Value Sources can be either a Key, a Constant or a value returned from an Internal or External Conversion Function.

It is recommended to build unique names by using keywords from the header(s) of the file to compose the names. Special characters will be translated into underscore characters (see also 2.2).

## 2.5 EXTENDED FITS FILE SPLITTING

The tool supports splitting up of FITS files using the FITS image extension. It is not possible in the present version to extract only a selection of extensions. I.e., all extensions are extracted and written into separate files. Each of these files is generated by merging the header of the extension 'on top of' the primary header.

The naming scheme applied is the same as for File Renaming (see 2.4).

## 2.6 EXTRACTING FITS HEADERS

While processing FITS files it is possible to have the FITS extracted and stored into ASCII files only containing the header information and no data. This can be done in two ways: One is to store the headers as in the FITS files, i.e. without a newline character at the end of each keyword card ('raw format'). The other option is to store it in a format whereby a newline character is contained at the end of each card in the 81st column ('readable format'). When storing in raw format, there might be blank cards in the HDUs in the extracted file. When storing in the readable format, blank keyword cards are removed.

The extracted header information is stored in files with the name "<target filename>.hdr", whereby the "<target filename>" is a name of a target file generated with extension ".hdr" rather than ".fits". The storage directory is the same as where the target file is located.

When requesting header extraction and file splitting at the same time, an extracted FITS header file will be generated for each extension, i.e. for each target file.

## 2.7 SPECIFYING OUTPUT FILE TARGET DIRECTORY

The files produced during operation of the FTU are default stored in the directory of the source file. Within the FTT however, it can be specified to use another directory as target directory. Note that this directory must exist, i.e., FTU will not try to create it.

A target directory and target filename can also be specified with input parameters (see 2.14).

## 2.8 CHANGES INTRODUCED TO FITS HEADERS

Empty HDU blocks (2880 bytes blocks containing blanks) will automatically be removed from the headers processed. In the present release it is not possible to disable this feature.

The FTU does not allow to have repeated keywords within one header. If several occurrences of one keyword are encountered, the last occurrence will be taken into account. Exceptions to this rule are HISTORY and COMMENT keyword cards.

By default the tool will add a COMMENT entry in the files translated, indicating that this file was translated with the tool, the date the translation was performed, and the FTT used. If not explicitly requested, the tool will not perform a translation on a file with the same FTT. This can be overridden by the input parameter “-force” (see section 2.14). The comment has the following appearance (example):

```
COMMENT FTU-1.0/2000-02-01T10:25:26/BitpixTT
```

For each action changing the header such that a keyword disappears from the header or the value changes, a HISTORY entry is added in the header concerned. This can be disabled as well by an input parameter (“-noHistory”). These entries have the format:

```
HISTORY FTU-<version>/<date>/<action>: <keyword> = <value>
```

E.g.:

```
HISTORY FTU-1.1/2000-02-07/REMOVE: ORIGIN = "ESO-LASILLA"
```

- indicating that version 1.1 of the tool removed the key ORIGIN which had the value “ESO-LASILLA”.

With these history entries it should be possible to more or less re-generate the original contents of a FITS file.

## 2.9 VALUE SOURCES/CONVERSION FUNCTIONS

As mentioned in a previous section, the values to be used e.g. for the Add Action can come from three different Value Sources: 1) Another key, 2) A Conversion Function and 3) A constant). An arbitrary number of Value Sources can be specified. It is however important to supply a Conversion Function to process the input values to yield one, final result. The sequence in which the Value Sources are specified is taken into account and the values fed into the Conversion Function in that order. A Conversion Function must deliver one value as return value. This value should be formatted as a string. The FTU will take care of re-formatting the value according to the specification in the Dictionary. The value of a logical parameter is given as “T” or “F”. If a space character is desired as value for a constant, the tag “\$BLANK” should be used.

A number of Internal Conversion Functions (ICFs) are provided within the FTU. However, in order to obtain flexibility, External Conversion Functions (ECFs) can be provided as well. This makes it possible for each user to provide his own set of Conversion Functions. These programs must be executable on the shell and must follow the rules described above when it comes to the input parameters and the return value.

In this release the following ICFs are provided

<b>ConcatString</b>	Cancatenate the strings given as input parameter.
<b>AndLogical</b>	And a list of logical values.

<b>OrLogical</b>	Or a list of logical values.
<b>SumInteger</b>	Sum up a list of integer values.
<b>MultiplyInteger</b>	Multiply a list of integer values.
<b>SumDouble</b>	Sum up a list of double values.
<b>MultiplyDouble</b>	Multiply a list of double values.
<b>Ut2IsoTime</b>	Converts a UT time (time since midnight) into a time string of the ISO8606 format (HH:MM:SS[.ss]).

This can be used e.g. to convert LST or UTC to a more readable format.

These are quite general, and maybe not very useful in most cases (with exception of MultiplyDouble which can be used to scale a floating point value). The idea is to extend the tool with more specific ICFs. Users are encouraged to propose additional ICFs if it is believed that these are of a general interest.

Context specific functions should be provided as ECFs. There is a penalty in terms of execution time using ECFs rather than ICFs. The latter is done by adding a "SHELL:" in front of the command to be executed on the UNIX shell. E.g.: "ADD.FCT SHELL:MyFct", will execute the command "MyFct" on the shell and assign the returned value to the keyword added.

FTU will check the exit code from the ECF. If this is equal to zero (0), the execution is considered as successful. If the exit code is different from zero, the execution of the ECF is considered to have failed and FTU will treat this as an error condition. I.e., if Tolerant Mode is specified it will skip this Translation, and carry on with the next. If Tolerant Mode is not specified, FTU will bail out with an error message.

A simple protocol has been defined, to allow ECFs to 'communicate' with the FTU. The protocol is based on the output value written by the ECF on stdout. By applying a special tag, the ECF can give 'instructions' to the FTU. This protocol is defined as follows:

```
<FTU ECF Com. Tag>: <info>|<cmd>
```

- where:

```
<FTU ECF Com. Tag> := #FTU-CMD | #FTU-ERR
```

#FTU-CMD: Instruct the FTU to carry out an action.

#FTU-ERR: The execution of the ECF failed. The FTU ECF Message will contain error diagnostics, which the FTU will feed back to the user or application that invoked the FTU to identify the error.

The contents of the ECF instruction are defined as:

```
<info> := <ASCII message>
```

```
<cmd> := REMKEY
```

REMKEY: Remove the key in connection with the Translation.

When using the FTU-ECF command interface, the exit code of the ECF should be zero (0).

More commands can be added if needed. The definition must be refined.

One ECF is provided with the FTU. This is a small tool called ftuGenMoonPars. It can be used to calculate various parameters in connection with the moon. The input parameters are as follows:

Correct usage is:

```
> ftuGetMoonPars -ra <pars> | -dec <pars> | -dist <pars> |
                  -alt <pars> | -az <pars> | -phase <pars>
```

Parameters:

-ra: <pars> := MJD-OBS TEL.GEOLAT LST

-dec: <pars> := MJD-OBS TEL.GEOLAT LST

-dist: <pars> := RA DEC GEN.MOON.RA GEN.MOON.DEC

-alt: <pars> := MJD-OBS TEL.GEOLAT LST

-az: <pars> := MJD-OBS TEL.GEOLAT LST

-phase: <pars> := MJD-OBS

## 2.10 TYPE CONVERSION

When a value of a certain type is allocated the value of another type a type conversion is taking place. An example of this is when a string value is copied (assigned) to a keyword of type double. The rules for this type conversion is described in the table below

Type	1: String	2: Logical	3: Integer	4: Double
1: String	'<str>' -> '<str>'	'T*' -> T 'F*' -> F '<str>' <sup>a</sup> -> F	'<int>*' -> <int> Result is rounded if necessary. '<str>' <sup>b</sup> -> 0	'<dbl>*' -> <dbl> '<str>' <sup>c</sup> -> 0.0
2: Logical	T -> 'T' ' F -> 'F' '	T -> T F -> F	T -> 1 F -> 0	T -> 1.0 F -> 0.0
3: Integer	<int> -> '<int>' <sup>d</sup>	<int> != 0 -> T <int> == 0 -> F	<int> -> <int>	<int> -> <dbl>
4: Double	<dbl> -> '<dbl>' <sup>e</sup>	<dbl> != 0 -> T <dbl> == 0 -> F	<dbl> -> <int> Result is rounded if necessary.	<dbl> -> <dbl>

- String not containing T or F as first character.
- String not containing an initial integer number ([<->digit][<digit>]).
- String not containing an initial double number.
- Integer converted to string.

- e. Double converted to string.

## 2.11 IMAGE STATISTICS

By default, the tool will generate statistics for each image contained in a FITS file. The statistics computed and the corresponding keywords are:

<b>DATAMIN</b>	The minimum pixel value encountered in the image.
<b>DATAMAX</b>	The maximum pixel value encountered in the image.
<b>DATAMEAN</b>	The mean value for the pixels in the image.
<b>DATAMED</b>	The median pixel value for the frame.
<b>DATARMS</b>	The root-mean-square for the pixel values.

The 5 keywords listed above will be added in the header for each image found in a FITS file.

The values for MEDIAN, RMS and MEAN are not estimated values, but are the actual computed ones. I.e., all pixel values are taken into account, and it is possible to rely on the correctness of these values.

## 2.12 CALCULATION OF CHECKSUM

By default the tool will calculate the checksum for a FITS file and write the CHECKSUM keyword in the primary header. The CHECKSUM is calculated for the entire FITS file and not for single extensions.

The standard ESO algorithm is used to compute the ASCII 1's complement CHECKSUM. Refer to <http://archive.eso.org/dicb> for further information.

## 2.13 LOGGING ACTIONS IN A LOG FILE

The tool can be requested to produce logs and to direct these to a file which must be specified with an input parameter (-logFile). If the log file already exists, the new logs generated will be added at the end of the file. Some care should be taken what concerns the size of this file if the same file is used for logging for a longer period of time, or when the logging level is high.

The format of the logs is as follows:

```
<ISO8601 time stamp> [<log type>] <log message>
```

The <log type> can be one of "INFO" for logs which are giving information about the actions performed by the tool, "WARNING" in case a recoverable problem was encountered, "SEVERE" if a non-recoverable problem occurred. The latter indicates a severe error and it might be that such an error influences the result of the translation. I.e., it might be that an action could not be carried out like adding a keyword to a header. The "FATAL" tag indicates a serious, non-recoverable error, which means that the translation has to be aborted.

An example of a log is:

```
2000-02-01T13:49:09.0751 [INFO] Registering file to be checked: /home/jknudstr/tmp/ISAAC.fits.
```

See section 2.14 for an explanation of the input parameters.

## 2.14 THE COMMAND LINE PARAMETERS

Online help about the input parameters to the fitsTranslate tool can be obtained by typing:

```
% fitsTranslate -h
```

on the UNIX shell.

A more detailed description of the input parameters is given here:

**[-v <level=1..5>]** Specifies if verbose output should be generated on stdout. The level indicates how detailed the information should be. Level 5 yields the most detailed information.

The verbose output is given in the format:

```
<source file>:<line no>:<time stamp>> <info>
```

- like e.g.:

```
ftuFTT.C:1340:2000-02-07T10:51:51.4429> Calculating median
...
```

**[-force]** This flag is used to force the tool to execute the translation on a file which contains the FTU Translation Comment indicating that the file has already been translated using that particular FTT.

**[-test]** Indicates for the tool that the input file(s) should not be overwritten. Instead the output filename will be the name of the source file(s) with a “\_test” appended to the filename. I.e., “<path>/<filename>\_test.fits” will be the format of the name.

This is used e.g. while developing an FTT to avoid overwriting the test file(s) used.

**[-noStat]** Indicates for the tool that no statistics should be generated.

Existing DATAMIN, DATAMAX, DATAMEAN, DATAMED and DATARMS keywords are not removed, but it cannot be guaranteed that the values are correct, as these are generated by another instance. If these keywords are already contained in the header, and the FTU generates the statistics, the old values will be overwritten.

**[-noChecksum]** Indicates to the tool that it should not calculate the checksum for the file. If the CHECKSUM keyword is already found in the primary header, it is removed as it will have a wrong value after the translation.

**[-noHistory]** If given, the tool will not generate HISTORY entries for the changes introduced as described in 2.8.

<b>[-noComment]</b>	If specified, the tool will not add the comment indicating that the file was translated with the specified FTT as described in section 2.8.
<b>[-skipFileOnError]</b>	The flag is used to force the tool to continue the translation of a batch of files even though the translation of one of the files had to be aborted.
<b>[-tolerant]</b>	Make the tool run in a more permissive mode, whereby it will try to continue to translate a file even though an error of type WARNING/SEVERE occurs. This means that possibly not all the requested translations have been carried out successfully.
<b>[-logFile &lt;filename&gt;]</b>	Specifies a log file in which to write the logging information produced during the translation. If the file does not exist, it is created, otherwise the logs will be appended to an existing file.
<b>[-logLevel &lt;level&gt;]</b>	Indicates the level used by the logging feature. Usually in a production system it should be enough with level 1 which more or less only indicates the names of the files that are being translated.  At the highest level, all the actions performed by the tool are logged, like for instance all the Add, Change, ... Actions performed.
<b>[-overwrite]</b>	Overwrite target FITS files existing. I.e., don't force unique, indexed names.
<b>[-targetDir &lt;dir&gt;]</b>	Specify a Target Directory in which to put the FITS files generated. Overwrites a possible Target Directory specified in the FTT (FILE.PATH.*).
<b>[-targetFile &lt;name&gt;]</b>	Specify a Target Filename to use to build the names of the FITS files generated. Overwrites possible settings in this connection in the FTT (FILE.RENAME.*, FILE.SPLIT.*).
<b>[-ftt &lt;ftt name&gt;]</b>	Defines the FTT to use for the translation. The file must be given with the full path if not stored in the same directory from where the tool is being executed. See 2.15 for a description of the FTT.
<b>&lt;file (list)&gt;</b>	List of FITS files to be translated. If not located in the same directory as where the tool is executed, the full path name should be specified.

As can be seen from the list above, only the file list is mandatory.

## 2.15 THE FITS TRANSLATION TABLE

The FTT is an ASCII file in the PAF format as specified in the "Data Interface Control Document", [2]. Please refer to this document for a description of the format. It is recommended to give the FTT files the extension ".ftt" although this is not enforced by the tool.

The FTT contains a list of actions that the FTU should carry out. The order in which these actions are specified, is the order in which they are executed. It is important to specify the sequence of actions with some care.

The tags available (the keywords) to define the translation, are defined in the following table:

Keyword Group	Explanation
INCLUDE	The INCLUDE key is used to specify another FITT which will be loaded into (merged together) with the present one being loaded. The insertion will be done at the place the INCLUDE key is encountered.
<i>[DICTIONARY]<sup>a</sup></i>	Indicates to the FTU a Dictionary that should be loaded. The filename should be given with the complete path. Note, if “ALL” is specified as a name of a dictionary, all DIDs found will be loaded.
FILE.PATH.BASE <i>[FILE.PATH.CONCAT.KEY]</i> <i>[FILE.PATH.CONCAT.CONST]</i> <i>[FILE.PATH.CONCAT.FCT]</i> <i>[FILE.PATH.CONCAT.KEY.HDR]</i>	Using these keys, it is possible to specify a target directory in which the target files are stored. These are e.g. the translated files, possibly renamed, and the files resulting from an extended FITS file splitting or extraction of FITS headers.  See also description for FILE.RENAME.PREFIX.
HDR.DUMP	If present, it indicate for the FTU to dump the contents of FITS headers into ASCII files. This can be done according to two different schemes: <ol style="list-style-type: none"> <li>1. No newline character added at the end of each keyword card: “NO_NEWLINE”.</li> <li>2. Newline character added at the end of each keyword card in the 81st column: “NEWLINE”.</li> </ol>
FILE.RENAME.PREFIX <i>[FILE.RENAME.CONCAT.KEY]</i> <i>[FILE.RENAME.CONCAT.CONST]</i> <i>[FILE.RENAME.CONCAT.FCT]</i> <i>[FILE.RENAME.CONCAT.KEY.HDR]</i>	Indicates that the tool should rename the file(s) being translated. The new filename is build by appending the values queried from the Value Sources to the prefix. The prefix may be empty (“”). If “\$FILENAME” is used as value for the prefix, the name of the input file will be used as prefix.  Any sequence and occurrence of the keywords “FILE.RENAME.CONCAT.KEY”, “FILE.RENAME.CONCAT.CONST” and “FILE.RENAME.CONCAT.FCT” is legal. Special characters will be transformed into underscore. The “FILE.RENAME.CONCAT.KEY.HDR” key, is used to specify from which header the key should be extracted.  File Renaming cannot be applied to a file in conjunction with File Splitting (see below).
FILE.SPLIT.PREFIX <i>[FILE.SPLIT.HDUS]</i> <i>[FILE.SPLIT.CONCAT.KEY]</i> <i>[FILE.SPLIT.CONCAT.CONST]</i> <i>[FILE.SPLIT.CONCAT.FCT]</i> <i>[FILE.SPLIT.CONCAT.KEY.HDR]</i>	Used to indicate to the tool to split up FITS image files using the FITS image extension.  The names are build in the same way as for File Renaming.  The FILE.SPLIT.HDUS key is used to define a selected number of HDUS to extract from a FITS file using the extension. The default behavior is to extract them all. The HDUs must be given as, e.g.: “1,3,6”.  Cannot be used in conjunction with File Renaming.

Keyword Group	Explanation
ADD.KEY ADD.HDR  <i>[ADD.VALUE.KEY]<sup>b</sup></i> <i>[ADD.VALUE.CONST]</i> <i>[ADD.VALUE.FCT]</i>  <i>[ADD.FCT]</i>	<p>Indicates that the key listed should be added in the header specified. Note, that the primary header has number 0, and the first extension 1, and so forth. The keyword should be given in the Short-FITS format, e.g., "HIERARCH ESO DET DIT" -&gt; "DET.DIT".</p> <p>With these keys a list of Value Sources can be specified. The sequence in which the Values Sources is specified is the sequence in which they will be given into the Conversion Function. When specifying more than one Value Source a Conversion Function must always be specified to convert these values into one value.</p> <p>Conversion Function to be applied on the values. If a Conversion Function is specified, there must be at least one Value Source defined. If the function is internal the name should be given as listed above (section 2.9). If the function is external, this is indicated by prefixing the name with "SHELL:", e.g. "SHELL:ConvertMyValue". The tool will then execute this function as a program on the shell. The program must be available within the paths specified in the PATH environment variable.</p> <p>Note, it is possible to add COMMENT keywords using the ADD.KEY Action. It works as when adding other keywords, but it is possible to have multiple occurrences of the COMMENT keyword in the FITS headers.</p>
CHANGE.KEY CHANGE.HDR  <i>[CHANGE.VALUE.KEY]</i> <i>[CHANGE.VALUE.CONST]</i> <i>[CHANGE.VALUE.FCT]</i>  <i>[CHANGE.FCT]</i>	<p>Indicates to the tool to change the name of a keyword in a specified header.</p> <p>See description for ADD.VALUE.*.</p> <p>See description for ADD.FCT.</p>
RENAME.SRCKEY RENAME.TRGKEY RENAME.HDR  <i>[RENAME.FCT]</i>	<p>Defines the properties for carrying out a renaming of a key. The source keyword name and target keyword and the header must be defined.</p> <p>See description of ADD.FCT above.</p>
MOVE.KEY MOVE.SRCHDR MOVE.TRGHDR  <i>[MOVE.FCT]</i>	<p>Indicates to the tool to carry out a Move Action on a key to move it to another header. The keyword, the source and target header must be defined.</p> <p>See description of ADD.FCT.</p>
COPY.SRCKEY COPY.SRCHDR COPY.TRGKEY COPY.TRGHDR  <i>[COPY.FCT]</i>	<p>Makes the tool perform a Copy Action of a keyword. The source key and header, and the target key and header must be given.</p> <p>See description of ADD.FCT.</p>

Keyword Group	Explanation
REMOVE.KEYLIST REMOVE.HDR	Indicates for the tool that a Remove Action should be carried out on a key or a number of keys specified in a list. The header from which the keywords should be removed must be given. Wildcards can be used to specify the keyword names. E.g.:  REMOVE.KEYLIST DET.VOLT.* ,CROTAL ,PC001001  - which means that all keys of category “DET” having the subsystem “VOLT”, are removed. In addition keywords with the name “CROTAL” and “PC001001” will be removed.
DIC.KEY DIC.TYPE DIC.FORMAT DIC.COMMENT	With these keywords it is possible to define the properties for keywords which are not defined in any Dictionary loaded (if a Dictionary is loaded at all).  The way to fill out these fields does not differ from a normal Dictionary. Refer to [2]/Chapter 8 for further explanation.

- a. Keywords in []’s are optional. Keyword in []’s and bold are optional, but can be repeated.  
b. For the Add Action element, at least one Value Source must be specified.

In the following an example of an FTT is given. Not all possible tags are used. Comments can be written in the files if initiated with a hash sign. The table may not make a lot of sense what concerns the actions performed, and serves merely as an example:

```
# Specify other TTs to include.
# The contents of the specified file will be loaded into this FTT at the
# place where the "INCLUDE" statement occurs.
INCLUDE          FtuTestTable2

# Specify Dictionaries.
# These Dictionaries are loaded into the tool. It is not mandatory to
# load Dictionaries, but new keywords being added should be defined
# either in a Dictionary or internally in the FTT.
DICTIONARY       ~/ROOTS/INS_ROOT/SYSTEM/Dictionary/ESO-VLT-DIC.IRD
DICTIONARY       ~/ROOTS/INS_ROOT/SYSTEM/Dictionary/ESO-VLT-DIC.ISAAC_OS

# Rename the file.
# The target filename will be something like:
# "ThisIsAPrefix.<val TELESCOP>_<val INSTRUME>_DET.DIT_<val DET.DIT>.fits".
FILE.RENAME.PREFIX      ThisIsAPrefix.
FILE.RENAME.CONCAT.KEY  TELESCOP
FILE.RENAME.CONCAT.KEY  INSTRUME
FILE.RENAME.CONCAT.CONST DET.DIT
FILE.RENAME.CONCAT.KEY  DET.DIT

# Example of a Change Element.
# This action redefines the value of the keyword "DEC" found in the first
# extension. It takes the current value of "DEC" and multiplies this with
# "-1" using the Internal Conversion Function "MultiplyDouble".
CHANGE.KEY          DEC
CHANGE.HDR          1
CHANGE.VALUE.KEY    DEC
```

```
CHANGE.VALUE.CONST -1
CHANGE.FCT          MultiplyDouble
```

```
# Example of an Add Element.
# This action will apply the Internal Conversion Function "SumDouble" to the
# value of the keyword "EXPTIME" and the constant value "-0.567" and add this
# in the primary header (= 0).
ADD.KEY             DET.TST.KEY
ADD.HDR             0
ADD.VALUE.KEY       EXPTIME
ADD.VALUE.CONST     -0.567
ADD.FCT             SumDouble
```

```
# Example of a Change Element.
# This action will set the value of the keyword "DET.NDIT" in the first
# extension to "54321".
CHANGE.KEY          DET.NDIT
CHANGE.HDR          1
CHANGE.VALUE.CONST  54321
```

```
# Example of a Rename Element.
# This action will rename the keyword "PI-COI" to "OBSERVER" in the primary
# header.
RENAME.SRCKEY       PI-COI
RENAME.TRGKEY       OBSERVER
RENAME.HDR          0
```

```
# Example of a Move Element.
# This action will move the keyword "DET.NCORRS" from the first extension
# header to the second and apply the External Conversion Routine
# "MyConvFunction" to the value before writing it in the header.
MOVE.KEY            DET.NCORRS
MOVE.SRCHDR         1
MOVE.TRGHDR         2
MOVE.FCT            SHELL:MyConvFunction
```

```
# Example of a Copy Element.
# This action will copy the keyword "OBS.TPLNO" from the primary header
# to the first extension header, applying the External Conversion Function
# "TplNo2ObsId" to the value.
COPY.SRCKEY         OBS.TPLNO
COPY.SRCHDR         0
COPY.TRGKEY         OBS.ID
COPY.TRGHDR         1
COPY.FCT            SHELL:TplNo2ObsId
```

```
# Example of a Remove Element.
# This action will remove all keywords of containing the pattern
# "DET.VOLT.*" from the primary header. In addition, the keywords "CROTAL"
# and "PC001001" will be removed.
REMOVE.KEYLIST      DET.VOLT.*,CROTAL,PC001001
REMOVE.HDR          1
```

```
# Define keywords.
# This is an example of an Internal Dictionary Keyword Definition. If keywords
# are already defined in a Dictionary loaded, it is not necessary to define
```

```
# them here. If this is done anyway, the definition in the FTT will overwrite
# the definition in the Dictionary. Keywords being added to a file, should
# be defined either in a Dictionary or in the FTT.
```

```
DIC.KEY          DET.TST.KEY
DIC.TYPE         double
DIC.FORMAT       %.4E
DIC.COMMENT      This is an example
```

```
# --- oOo ---
```

## 2.16 CAUTIONS

The section contains a list of aspects where caution should be applied.

### 2.16.1 ‘Problematic’ Keywords

Even though the tool can work without the availability of Dictionary definitions for keywords, and that this normally does not cause any problems, there might be some special cases where the result of keyword cards in the target file is not as expected. This is caused by the fact that the FTU makes assumptions concerning the keywords when it comes to type and format when the Dictionary definition for a keyword is not available. To avoid this situation, a proper Dictionary definition should be provided for these ‘problematic’ keywords.

Note that one particular case that cannot be handled properly by the tool, is when a keyword is using the value conversion feature to place a transformed representation of the value in the comment. In this case the comment of the target keyword remains the same even though changes were introduced to the value. In a case like this the proper Dictionary definition specifying the conversion tag in the comment, must be provided either in an external Dictionary or internally in the FTT.

When developing a FTT for a specific type of FITS file, it is recommendable to check thoroughly that the contents of the target file is as expected.

### 2.16.2 Keyword Definitions for Rename/Copy Actions

In the present version of the FTU, Dictionary definitions for all new keys added in the header(s) must be available. This also applies for the Rename and Copy Actions. If not available an error will be reported.

### 3 FTU FTT Python API

Even though the FTT format as defined presently, provides quite a wide variety of possibilities for performing various types of Translation Actions, also because of the concept of External Plug-In Conversion Functions, it is still a quite static definition with a lot of limitations. In addition, the use of External Plug-In Conversion Function may be a bit cumbersome. Furthermore, in many cases FTTs will grow to become quite large, if the same set of Translation Actions are carried on all extensions in a FITS file using the FITS Extension. In order to overcome this problem, the FTU FTT Python API is provided, in the form of a set of classes and a Python module with definitions.

While a normal FTT is a pretty static definition of the translations to carry out, 'dynamic' FTTs can be programmed using the FTT API, increasing the flexibility and usability of the FTU.

#### 3.1 CLASSES/MODULES IN THE FTT API

The following classes and modules are provided by the API:

Class/Module	Description
<b>FtuFtt</b>	This is the main class used to handle the parameters of an FTT. Methods are provided to load and save FTTs. It is possible to add and remove the various building blocks of the FTT from within this class. It is also possible to execute the FTT on a set of given files.
<b>FtuFttDefs</b>	<p>This module contains various definitions used when programming Python FTTs.</p> <p>The following 'constants' are defined for referring to Translation Actions:</p> <pre>ftuACTION_UNDEF, ftuACTION_HDR_DUMP, ftuACTION_FILE_PATH, ftuACTION_FILE_RENAME, ftuACTION_FILE_SPLIT, ftuACTION_ADD, ftuACTION_CHANGE, ftuACTION_RENAME, ftuACTION_MOVE, ftuACTION_COPY, ftuACTION_REMOVE</pre> <p>The following 'constants' are defined for referring to Value Sources:</p> <pre>ftuVAL_SRC_UNDEF, ftuVAL_SRC_KEY, ftuVAL_SRC_CONST, ftuVAL_SRC_FCT</pre> <p>The following 'constants' are defined for referring to Dictionary Types:</p> <pre>ftuTYPE_STRING, ftuTYPE_LOGICAL, ftuTYPE_INTEGER, ftuTYPE_DOUBLE</pre> <p>It is recommended to use these definitions of the various properties, and to avoid hard coding these into the code. It is probably most convenient to make these definitions available importing the FtuFttDefs module in the following way:</p> <pre>import from FtuFttDefs *</pre> <p>- to make the name space of FtuFttDefs available within the script.</p>

Class/Module	Description
<b>FtuFttDicRec</b>	Class used to handle the contents of an FTT Dictionary Record.
<b>FtuFttValueSrc</b>	Class used to handle the information of an FTT Value Source indicating from where to take a value.
<b>FtuFttAction</b>	This class is used to handle the Translation Actions: Add, Change, Rename, Move, Copy and Remove.
<b>FtuFttFilePath</b>	Class to handle the parameters in connection with a File Path Definition.
<b>FtuFttFileRename</b>	Class to handle the parameters in connection with a File Rename Definition.
<b>FtuFttFileSplit</b>	Class to handle the parameters in connection with a File Split Definition

The complete documentation for the API is available on-line via the FTU home page: <http://archive.eso.org/ftu>.

### 3.2 EXAMPLE OF A Python FTT

In the following a simple Python script is shown, which builds up a simple FTT. The example is based on a real case whereby FITS file from the VLT instrument VIMOS are translated (keywords changed, and extended FITS file split), before sending the files into the Archive System.

The source code for building up the FTT is as follows:

```
#!/opsw/util/python/bin/python
#*****
# ESO/DFS
#
# "@(#) $Id: PythonFttEx1.py,v 1.3 2000/11/14 07:38:41 jknudstr Exp $"
#
# Who          When          What
# -----
# jknudstr    20/02/2001    Created
#

import os, sys
# To be able to use the 'constants' referring to Translation Actions etc.
from FtuFttDefs import *
import FtuFtt, FtuFttAction, FtuFttValueSrc

# We take a FITS to convert as input parameter.
if (len(sys.argv) != 2):
    print "\n\nCorrect usage is:\n\n"
    print "% PythonFttEx1 <FITS file>\n\n"
    sys.exit(1)

# Create instance of the FTT class.
ftt = FtuFtt.FtuFtt()

# Add Remove Actions.
for hdrNo in [1, 2]:
    ftt.addAction(FtuFttAction.FtuFttAction().\
        setRemoveAction("XTENSION,EXTNAME,ORIGIN,"+
```

```

                                "DET.CHIP.XGAP,DET.CHIP.YGAP",hdrNo))
for hdrNo in [1, 2]:
    ftt.addAction(FtuFttAction.FtuFttAction().\
                  setRemoveAction("INS.SLIT*.*", hdrNo))

# Add Change Actions.
for valSet in [{"DET.CHIPS", 1,
                [[ftuACTION_CHANGE, ftuVAL_SRC_CONST, "1"]], ""},
               ["DET.OUTPUTS", 1,
                [[ftuACTION_CHANGE, ftuVAL_SRC_CONST, "1"]], ""},
               ["DET.WIN1.NX", 1,
                [[ftuACTION_CHANGE, ftuVAL_SRC_KEY, "DET.WIN1.NX"],
                 [ftuACTION_CHANGE, ftuVAL_SRC_CONST, "0.5"]],
                "MultiplyDouble"],
               ["DET.CHIPS", 2,
                [[ftuACTION_CHANGE, ftuVAL_SRC_CONST, "1"]], ""},
               ["DET.OUTPUTS", 2,
                [[ftuACTION_CHANGE, ftuVAL_SRC_CONST, "1"]], ""},
               ["DET.OUT.X", 2,
                [[ftuACTION_CHANGE, ftuVAL_SRC_CONST, "1"]], ""},
               ["DET.WIN1.NX", 2,
                [[ftuACTION_CHANGE, ftuVAL_SRC_KEY, "DET.WIN1.NX"],
                 [ftuACTION_CHANGE, ftuVAL_SRC_CONST, "0.5"]],
                "MultiplyDouble"]]:
    valSrcList = []
    for valSrc in valSet[2]:
        valSrcList.append(FtuFttValueSrc.\
                          FtuFttValueSrc(valSrc[0], valSrc[1], valSrc[2]))
    ftt.addAction(FtuFttAction.FtuFttAction().\
                  setChangeAction(valSet[0], valSet[1], valSrcList, valSet[3]))

# Set File Split Action.
valSrcs = [FtuFttValueSrc.FtuFttValueSrc(ftuACTION_FILE_SPLIT,
                                          ftuVAL_SRC_CONST, "."),
           FtuFttValueSrc.FtuFttValueSrc(ftuACTION_FILE_SPLIT,
                                          ftuVAL_SRC_KEY, "OCS.CON.QUAD")]
ftt.getFileSplitAction().setFileSplit("$FILENAME", valSrcs)

# Execute the FTT.
ftt.execute(sys.argv[1])

#
# ____oOo____

```

There are of course many different ways in which this could be implemented when it comes to the Python details. This is up to the individual developer to use the features of Python in the best suitable way.

The resulting FTT has the following contents:

FILE.SPLIT.PREFIX	"\$FILENAME"
FILE.SPLIT.CONCAT.CONST	"."
FILE.SPLIT.CONCAT.KEY	"OCS.CON.QUAD"
REMOVE.KEYLIST	"XTENSION,EXTNAME,ORIGIN,DET.CHIP.XGAP,DET.CHIP.YGAP"
REMOVE.HDR	1
REMOVE.KEYLIST	"XTENSION,EXTNAME,ORIGIN,DET.CHIP.XGAP,DET.CHIP.YGAP"
REMOVE.HDR	2

```

REMOVE.KEYLIST      "INS.SLIT*.*"
REMOVE.HDR           1

REMOVE.KEYLIST      "INS.SLIT*.*"
REMOVE.HDR           2

CHANGE.KEY           "DET.CHIPS"
CHANGE.HDR           1
CHANGE.VALUE.CONST   "1"

CHANGE.KEY           "DET.OUTPUTS"
CHANGE.HDR           1
CHANGE.VALUE.CONST   "1"

CHANGE.KEY           "DET.WIN1.NX"
CHANGE.HDR           1
CHANGE.VALUE.KEY      "DET.WIN1.NX"
CHANGE.VALUE.CONST    "0.5"
CHANGE.FCT           "MultiplyDouble"

CHANGE.KEY           "DET.CHIPS"
CHANGE.HDR           2
CHANGE.VALUE.CONST    "1"

CHANGE.KEY           "DET.OUTPUTS"
CHANGE.HDR           2
CHANGE.VALUE.CONST    "1"

CHANGE.KEY           "DET.OUT.X"
CHANGE.HDR           2
CHANGE.VALUE.CONST    "1"

CHANGE.KEY           "DET.WIN1.NX"
CHANGE.HDR           2
CHANGE.VALUE.KEY      "DET.WIN1.NX"
CHANGE.VALUE.CONST    "0.5"
CHANGE.FCT           "MultiplyDouble"

# --- oOo ---

```

The example is not very good for demonstrating how the handling of the FTT can be simplified, but illustrates how the Python code statements transform into an FTT.